METHODS AND APPARATUS FOR EFFICIENT COMPLEX LONG MULTIPLICATION AND COVARIANCE MATRIX IMPLEMENTATION

Related Applications

The present application claims the benefit of U.S. Provisional Application Serial No. 60/244,861 entitled "Methods and Apparatus for Efficient Complex Long Multiplication and Covariance Matrix Implementation" and filed November 1, 2000, which is incorporated by reference herein in its entirety.

Field of the Invention

The present invention relates generally to improvements to parallel processing, and more particularly to methods and apparatus for efficiently calculating the result of a long complex multiplication. Additionally, the present invention relates to the advantageous use of this approach for the calculation of a covariance matrix.

Background of the Invention

The product of two complex numbers x and y is defined to be $z = x_R y_R - x_I y_I + i(x_R y_I + x_I y_R)$, where $x = x_R + ix_I$, $y = y_R + iy_I$ and i is an imaginary number, or the square root of negative one, with $i^2 = -1$. This complex multiplication of x and y is calculated in a variety of contexts, and it has been recognized that it will be highly advantageous to perform this calculation faster and more efficiently.

Summary of the Invention

The present invention defines hardware instructions to calculate the product of two complex numbers encoded as a pair of two fixed-point numbers of 16 bits each. The product may be calculated in two cycles with single cycle pipeline throughput efficiency, or in a single cycle. The product is encoded as a 32 bit real component and a 32 bit imaginary component. The present invention also defines a series of multiply complex instructions with an accumulate operation. Additionally, the present invention also defines a series of multiply complex instructions with an extended precision accumulate operation. The complex long instructions and methods of the present invention may be advantageously used in a variety of

contexts, including calculation of a fast Fourier transform as addressed in U.S. Patent Application Serial No. 09/337,839 filed June 22, 1999 entitled "Efficient Complex Multiplication and Fast Fourier Transform (FFT) Implementation on the ManArray Architecture" which is incorporated by reference herein in its entirety. The multiply complex instructions of the present invention may be advantageously used in the computation of a covariance matrix, as described below.

A more complete understanding of the present invention, as well as other features and advantages of the invention will be apparent from the following Detailed Description and the accompanying drawings.

Brief Description of the Drawings

Fig. 1 illustrates an exemplary 2x2 ManArray iVLIW processor;

Fig. 2A illustrates a multiply complex long (MPYCXL) instruction in accordance with the present invention;

Figs. 2B and 2C illustrate the syntax and operation of the MPYCXL instruction of Fig. 2A;

Fig. 3A illustrates a multiply complex conjugate long (MPYCXJL) instruction in accordance with the present invention;

Figs. 3B and 3C illustrate the syntax and operation of the MPYCXJL instruction of Fig. 3A;

Fig. 4A illustrates a multiply complex long accumulate (MPYCXLA) instruction in accordance with the present invention;

Figs. 4B and 4C illustrate the syntax and operation of the MPYCXLA instruction of Fig. 4A;

Fig. 5A illustrates a multiply complex conjugate long accumulate (MPYCXJLA) instruction in accordance with the present invention;

Figs. 5B and 5C illustrate the syntax and operation of the MPYCXJLA instruction of Fig. 5A;

Fig. 6A illustrates a multiply complex long extended precision accumulate (MPYCXLXA) instruction in accordance with the present invention;

Figs. 6B and 6C illustrate the syntax and operation of the MPYCXLXA instruction of Fig. 6A;

Fig. 7A illustrates a multiply complex conjugate long extended precision accumulate (MPYCXJLXA) instruction in accordance with the present invention;

Figs. 7B and 7C illustrates the syntax and operation of the MPYCXJLXA instruction of Fig. 7A;

Fig. 8 shows a block diagram illustrating various aspects of hardware suitable for performing the MPYCXL, MPYCXJL, MPYCXLA, MPYCXJLA, MPYCXJLA, MPYCXLXA and MPYCXJLXA instructions in two cycles of operation in accordance with the present invention;

Fig. 9 shows an integrated product adder and accumulator in accordance with the present invention;

Fig. 10 shows a block diagram illustrating various aspects of hardware suitable for performing the MPYCXL, MPYCXJL, MPYCXLA, MPYCXJLA, MPYCXJLA, MPYCXLXA and MPYCXJLXA instructions in a single cycle of operation in accordance with the present invention; and

Figs. 11A-11I illustrate the calculation of a covariance matrix on a 2x2 processing array in accordance with the present invention.

Detailed Description

Further details of a presently preferred ManArray core, architecture, and instructions for use in conjunction with the present invention are found in: U.S. Patent Application Serial No. 08/885,310 filed June 30, 1997, now U.S. Patent No. 6,023,753, U.S. Patent Application Serial No. 08/949,122 filed October 10, 1997, now U.S. Patent No. 6,167,502, U.S. Patent Application Serial No. 09/169,256 filed October 9, 1998, now U.S. Patent No. 6,167,501, U.S. Patent Application Serial No. 09/169,072 filed October 9, 1998, now U.S. Patent No. 6,219,776, U.S. Patent Application Serial No. 09/187,539 filed November 6, 1998, now U.S. Patent No. 6,151,668, U.S. Patent Application Serial No. 09/205,558 filed December 4, 1998, now U.S. Patent No. 6,173,389, U.S. Patent Application Serial No. 09/215,081 filed December 18, 1998, now U.S. Patent No. 6,101,592, U.S. Patent Application Serial No. 09/228,374 filed January 12, 1999, now U.S. Patent No. 6,216,223, U.S. Patent Application Serial No. 09/471,217 filed December 23, 1999, now U.S. Patent No. 6,260,082, U.S. Patent Application Serial No. 09/472,372 filed December 23, 1999, now U.S. Patent No. 6,256,683, U.S. Patent Application Serial No. 09/238,446 filed January 28, 1999, U.S. Patent Application Serial No. 09/267,570 filed March 12, 1999, U.S. Patent Application Serial No. 09/337,839 filed June 22, 1999, U.S. Patent Application Serial No. 09/350,191 filed July 9, 1999, U.S. Patent Application Serial No. 09/422,015 filed October 21, 1999, U.S. Patent Application Serial No. 09/432,705 filed November 2, 1999, U.S. Patent Application Serial

No. 09/596,103 filed June 16, 2000, U.S. Patent Application Serial No. 09/598,567 filed June 21, 2000, U.S. Patent Application Serial No. 09/598,564 filed June 21, 2000, U.S. Patent Application Serial No. 09/598,566 filed June 21, 2000, U.S. Patent Application Serial No. 09/598,558 filed June 21, 2000, U.S. Patent Application Serial No. 09/598,084 filed June 21, 2000, U.S. Patent Application Serial No. 09/599,980 filed June 22, 2000, U.S. Patent Application Serial No. 09/711,218 filed November 9, 2000, U.S. Patent Application Serial No. 09/747,056 filed December 12, 2000, U.S. Patent Application Serial No. 09/853,989 filed May 11, 2001, U.S. Patent Application Serial No. 09/886,855 filed June 21, 2001, U.S. Patent Application Serial No. 09/791,940 filed February 23, 2001, U.S. Patent Application Serial No. 09/792,819 filed February 23, 2001, U.S. Patent Application Serial No. 09/792,256 filed February 23, 2001, U.S. Patent Application Serial No. _____ entitled "Methods and Apparatus for Efficient Vocoder Implementations" filed October 19, 2001, Provisional Application Serial No. 60/251,072 filed December 4, 2000, Provisional Application Serial No. 60/281,523 filed April 4, 2001, Provisional Application Serial No. 60/283,582 filed April 13, 2001, Provisional Application Serial No. 60/287,270 filed April 27, 2001, Provisional Application Serial No. 60/288,965 filed May 4, 2001, Provisional Application Serial No. 60/298,624 filed June 15, 2001, Provisional Application Serial No. 60/298,695 filed June 15, 2001, Provisional Application Serial No. 60/298,696 filed June 15, 2001, Provisional Application Serial No. 60/318,745 filed September 11, 2001, Provisional Application Serial No. _____ entitled "Methods and Apparatus for Video Coding" filed October 30, 2001, Provisional Application Serial No. _____ entitled "Methods and Apparatus for a Bit Rate Instruction" filed November 1, 2001, all of which are assigned to the assignee of the present invention and incorporated by reference herein in their entirety.

In a presently preferred embodiment of the present invention, a ManArray 2x2 iVLIW single instruction multiple data stream (SIMD) processor 100 shown in Fig. 1 contains a controller sequence processor (SP) combined with processing element-0 (PE0) SP/PE0 101, as described in further detail in U.S. Application Serial No. 09/169,072 entitled "Methods and Apparatus for Dynamically Merging an Array Controller with an Array Processing Element". Three additional PEs 151, 153, and 155 are also utilized. It is noted that the PEs can be also labeled with their matrix positions as shown in parentheses for PE0 (PE00) 101, PE1 (PE01)151, PE2 (PE10) 153, and PE3 (PE11) 155.

The SP/PE0 101 contains a fetch controller 103 to allow the fetching of short instruction words (SIWs) from a 32-bit instruction memory 105. The fetch controller 103 provides the typical functions needed in a programmable processor such as a program counter

(PC), branch capability, digital signal processing, EP loop operations, support for interrupts, and also provides the instruction memory management control which could include an instruction cache if needed by an application. In addition, the SIW I-Fetch controller 103 dispatches 32-bit SIWs to the other PEs in the system by means of a 32-bit instruction bus 102.

In this exemplary system, common elements are used throughout to simplify the explanation, though actual implementations are not so limited. For example, the execution units 131 in the combined SP/PE0 101 can be separated into a set of execution units optimized for the control function, e.g. fixed point execution units, and the PE0 as well as the other PEs 151, 153 and 155 can be optimized for a floating point application. For the purposes of this description, it is assumed that the execution units 131 are of the same type in the SP/PE0 and the other PEs. In a similar manner, SP/PE0 and the other PEs use a five instruction slot iVLIW architecture which contains a very long instruction word memory (VIM) memory 109 and an instruction decode and VIM controller function unit 107 which receives instructions as dispatched from the SP/PE0's I-Fetch unit 103 and generates the VIM addresses-and-control signals 108 required to access the iVLIWs stored in the VIM. These iVLIWs are identified by the letters SLAMD in VIM 109. The loading of the iVLIWs is described in further detail in U.S. Patent Application Serial No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW PE-to-PE Communication". Also contained in the SP/PE0 and the other PEs is a common PE configurable register file 127 which is described in further detail in U.S. Patent Application Serial No. 09/169,255 entitled "Methods and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision".

Due to the combined nature of the SP/PE0, the data memory interface controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The SP/PE0 controller 125 also is the source of the data that is sent over the 32-bit broadcast data bus 126. The other PEs 151, 153, and 155 contain common physical data memory units 123', 123", and 123"' though the data stored in them is generally different as required by the local processing done on each PE. The interface to these PE data memories is also a common design in PEs 1, 2, and 3 and indicated by PE local memory and data bus interface logic 157, 157' and 157". Interconnecting the PEs for data transfer communications is the cluster switch 171 more completely described in U.S. Patent Application Serial No. 08/885,310 entitled "Manifold Array Processor", U.S. Application Serial No. 09/949,122 entitled "Methods and Apparatus for Manifold Array

Processing", and U.S. Application Serial No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-to-PE Switch Control". The interface to a host processor, other peripheral devices, and/or external memory can be done in many ways. The primary mechanism shown for completeness is contained in a direct memory access (DMA) control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories via the multiplexed bus interface represented by line 185. A high level view of a ManArray Control Bus (MCB) 191 is also shown.

All of the above noted patents are assigned to the assignee of the present invention and incorporated herein by reference in their entirety.

Turning now to specific details of the ManArray processor as adapted by the present invention, the present invention defines the following special hardware instructions that execute in each multiply accumulate unit (MAU), one of the execution units 131 of Fig. 1 and in each PE, to handle the multiplication of complex numbers.

Fig. 2A shows a multiply complex long (MPYCXL) instruction 200 for the multiplication of two complex numbers in accordance with the present invention. The syntax and operation description 210 of the MPYCXL instruction 200 are shown in Figs. 2B and 2C. As seen in diagram 220 of Fig. 2C, the MPYCXL instruction 200 provides for the multiplication of two complex numbers stored in source register Rx and source register Ry. In step 222, the complex numbers to be multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 224, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 226, the products are subtracted and added in the form of (Xr*Yr) – (Xi*Yi) and (Xr*Yi) + (Xi*Yr). In step 228, the final result is written back to the target registers at the end of an operation cycle of the MPYCXL instruction 200 with a 32-bit real component and a 32-bit imaginary component placed in the target registers such that Rto contains the 32-bit real component and Rte contains the 32-bit imaginary component.

Fig. 3A shows a multiply complex conjugate long (MPYCXJL) instruction 300 for the multiplication of a first complex number and the conjugate of a second complex number in accordance with the present invention. The syntax and operation description 310 of the MPYCXJL instruction 300 are shown in Figs. 3B and 3C. As seen in diagram 320 of Fig. 3C, the MPYCXJL instruction 300 provides for the multiplication of two complex numbers

stored in source register Rx and source register Ry. In step 322, the complex numbers to be multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 324, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 326, the products are subtracted and added in the form of (Xr*Yr) + (Xi*Yi) and (Xi*Yr) - (Xr*Yi). In step 328, the final result is written back to the target registers at the end of an operation cycle of the MPYCXJL instruction 300 with a 32-bit real component and a 32-bit imaginary component placed in the target registers such that Rto contains the 32-bit real component and Rte contains the 32-bit imaginary component.

Fig. 4A shows a multiply complex long accumulate (MPYCXLA) instruction 400 for the multiplication of two complex numbers to form a product which is accumulated with the contents of target registers in accordance with the present invention. The syntax and operation description 410 of the MPYCXLA instruction 400 are shown in Figs. 4B and 4C. As seen in diagram 420 of Fig. 4C, the MPYCXLA instruction 400 provides for the multiplication of two complex numbers stored in source register Rx and source register Ry. In step 422, the complex numbers to be multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 424, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 426, the products are subtracted and added in the form of (Xr*Yr) – (Xi*Yi) and (Xr*Yi) + (Xi*Yr). In step 428, (Xr*Yr) – (Xi*Yi) is added to the contents of target register Rto and (Xr*Yi) + (Xi*Yr) is added, or accumulated, to the contents of target register Rte. The final result is written back to the target registers at the end of an operation cycle of the MPYCXLA instruction 400 with a 32-bit real component and a 32-bit imaginary component placed in the target registers such that Rto contains the 32-bit real component and Rte contains the 32-bit imaginary component. For a two cycle embodiment, the target registers are fetched on a second cycle of execution to allow repetitive pipelining to a single accumulation register even-odd pair.

Fig. 5A shows a multiply complex conjugate long accumulate (MPYCXJLA) instruction 500 for the multiplication of a first complex number and the conjugate of a second complex number to form a product which is accumulated with the contents of target registers in accordance with the present invention. The syntax and operation description 510 of the MPYCXJLA instruction 500 are shown in Figs. 5B and 5C. As seen in diagram 520 of Fig. 5C, the MPYCXJLA instruction 500 provides for the multiplication of two complex numbers stored in source register Rx and source register Ry. In step 522, the complex numbers to be

multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 524, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 526, the products are added and subtracted in the form of (Xr*Yr) + (Xi*Yi) and (Xi*Yr) - (Xr*Yi). In step 528, (Xr*Yr) + (Xi*Yi) is added, or accumulated, to the contents of target register Rto and (Xi*Yr) - (Xr*Yi) is added to the contents of target register Rte. The final result is written back to the target registers at the end of an operation cycle of the MPYCXJLA instruction 500 with a 32-bit real component and a 32-bit imaginary component placed in the target registers such that Rto contains the 32-bit real component and Rte contains the 32-bit imaginary component. For a two cycle embodiment, the target registers are fetched on the second cycle of execution to allow repetitive pipelining to a single accumulation register even-odd pair.

Fig. 6A shows a multiply complex long extended precision accumulate (MPYCXLXA) instruction 600 for the multiplication of two complex numbers to form a product which is accumulated with the contents of the extended precision target registers in accordance with the present invention. The syntax and operation description 610 of the MPYCXLXA instruction 600 are shown in Figs. 6B and 6C. As seen in diagram 620 of Fig. 6C, the MPYCXLXA instruction 600 provides for the multiplication of two complex numbers stored in source register Rx and source register Ry. In step 622, the complex numbers to be multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 624, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 626, the products are subtracted and added in the form of (Xr*Yr) – (Xi*Yi) and (Xr*Yi) + (Xi*Yr). In step 628, the 32-bit value (Xr*Yr) – (Xi*Yi) is added to the contents of the extended precision target register XPRBo||Rto and the 32-bit value (Xr*Yi) + (Xi*Yr) is added to the contents of the extended precision target register XPRBe||Rte. The final result is written back to the extended precision target registers at the end of an operation cycle of the MPYCXLXA instruction 600 with a 40-bit real component and a 40-bit imaginary component placed in the target registers such that XPRBo||Rto contains the 40-bit real component and XPRBe||Rte contains the 40-bit imaginary component. For a two cycle embodiment, the target registers are fetched on the second cycle of execution to allow repetitive pipelining to a single accumulation register even-odd pair.

The extended precision bits for the 40-bit results are provided by the extended precision register (XPR). The specific sub-registers used in an extended precision operation depend on the size of the accumulation (dual 40-bit or single 80-bit) and on the target CRF register pair specified in the instruction. For dual 40-bit accumulation, the 8-bit extension registers XPR.B0 and XPR.B1 (or XPR.B2 and XPR.B3) are associated with a pair of CRF registers. For single 80-bit accumulation, the 16-bit extension register XPR.H0 (or XPR.H1) is associated with a pair of CRF registers. During the dual 40-bit accumulation, the even target register is extended using XPR.B0 or XPR.B2, and the odd target register is extended using XPR.B1 or XPR.B3. The tables 602, 604, 608, 612 and 614 of Fig. 6A illustrate the register usage in detail.

As shown in Fig. 6A, the XPR byte that is used depends on the Rte. Further details of an XPR register suitable for use with the present invention are provided in U.S. Patent Application No. 09/599,980 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed on June 20, 2000 which is incorporated by reference herein in its entirety.

Fig. 7A shows a multiply complex conjugate long extended precision accumulate (MPYCXJLXA) instruction 700 for the multiplication of a first complex number and the conjugate of a second complex number to form a product which is accumulated with the contents of the extended precision target registers in accordance with the present invention. The syntax and operation description 710 of the MPYCXJLXA instruction 700 are shown in Figs. 7B and 7C. As seen in diagram 720 of Fig. 7C, the MPYCXJLXA instruction 700 provides for the multiplication of two complex numbers stored in source register Rx and source register Ry. In step 722, the complex numbers to be multiplied are organized in the source registers such that H1 contains the real component of the complex numbers and H0 contains the imaginary component of the complex numbers. In step 724, the complex numbers are multiplied to produce the products Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi. Next, in step 726, the products are subtracted and added in the form of (Xr*Yr) + (Xi*Yi) and (Xi*Yr) - (Xr*Yi). In step 728, the 32-bit value (Xr*Yr) + (Xi*Yi) is added to the contents of the extended precision target register XPRBe||Rte and the 32-bit value (Xi*Yr) - (Xr*Yi) is added to the contents of the extended precision target register XPRBo||Rto. The final result is written back to the extended precision target registers at the end of an operation cycle of the MPYCXJLXA instruction 700 with a 40-bit real component and a 40-bit imaginary component placed in the target registers such that XPRBo||Rto contains the 40-bit real component and XPRBe||Rte contains the 40-bit imaginary component. For a two cycle

embodiment, the target registers are fetched on the second cycle of execution to allow repetitive pipelining to a single accumulation register even-odd pair.

The extended precision bits for the 40-bit results are provided by the extended precision register (XPR). The specific sub-registers used in an extended precision operation depend on the size of the accumulation (dual 40-bit or single 80-bit) and on the target CRF register pair specified in the instruction. For dual 40-bit accumulation, the 8-bit extension registers XPR.B0 and XPR.B1 (or XPR.B2 and XPR.B3) are associated with a pair of CRF registers. For single 80-bit accumulation, the 16-bit extension register XPR.H0 (or XPR.H1) is associated with a pair of CRF registers. During the dual 40-bit accumulation, the even target register is extended using XPR.B0 or XPR.B2, and the odd target register is extended using XPR.B1 or XPR.B3. The tables 702, 704, 708, 712 and 714 of Fig. 7A illustrate the register usage in detail. As shown in Fig. 7A, the XPR byte that is used depends on the Rte.

All of the above instructions 200, 300, 400, 500, 600 and 700 may complete in 2 cycles and are pipelineable. That is, another operation can start executing on the execution unit after the first cycle. In accordance with another aspect of the present invention, all of the above instructions 200, 300, 400, 500, 600 and 700 may complete in a single cycle.

Fig. 8 shows a high level view of a hardware apparatus 800 suitable for implementing the multiply complex instructions for execution in two cycles of operation. This hardware capability may be advantageously embedded in the ManArray multiply accumulate unit (MAU), one of the execution units 131 of Fig. 1 and in each PE, along with other hardware capability supporting other MAU instructions. As a pipelined operation, the first execute cycle begins with a read of source register operands Ry.H1, Ry.H0, Rx.H1 and Rx.H0 from the compute register file (CRF) shown as registers 803 and 805 in Fig. 8 and as registers 111, 127, 127', 127", and 127"' in Fig. 1. These operands may be viewed as corresponding to the operands Yr, Yi, Xr and Xi described above. The operand values are input to multipliers 807, 809, 811 and 813 after passing through multiplexer 815 which aligns the halfword operands.

Multipliers 807 and 809 are used as 16 x 16 multipliers for these complex multiplications. The 32 x 16 notation indicates these two multipliers are also used to support 32 x 32 multiplies for other instructions in the instruction set architecture (ISA). Multiplexer 815 is controlled by an input control signal 817. The outputs of the multipliers, Xr*Yr, Xr*Yi, Xi*Yr and Xi*Yi, are input to registers 824a, 824b, 824c and 824d after passing through multiplexer 823 which aligns the outputs based on the type of multiplication operation. The registers 824a, 824b, 824c and 824d latch the multiplier outputs, allowing pipelined operation of a second instruction to begin. An output control signal 825 controls

the routing of the multiplier outputs to the input registers 824a, b, c, d of adders 819 and 821. The second execute cycle, which can occur while a new multiply complex instruction is using the first cycle execute facilities, begins with adders 819 and 821 operating on the contents of registers 824a, 824b, 824c and 824d. The adders 819 and 821 function as either adders or subtractors based on a conjugate select signal 827, which is set depending on the type of complex multiplication being executed.

The outputs of the adders 819 and 821 are then passed to accumulators 833 and 835. If an accumulate operation is not being performed, a zero value is output from multiplexers 829 and 831 to accumulators 833 and 835 to produce a zero input for no accumulation. If an accumulate operation is being performed, the contents of current target registers Rt.H1 and Rt.H1, shown as registers 837 and 839, is output from multiplexers 829 and 831 to accumulators 833 and 835 as an input to produce an accumulated result. Multiplexers 829 and 831 are controlled by an accumulator control signal 841. The outputs of the accumulators 823 and 825 are then written to the target registers 837 and 839 which contain the 32 bit real result and the 32 bit imaginary result, respectively.

If an extended precision operation is being performed, the accumulation is augmented eight extra bits by adding the contents of an extended precision registers 843 and 844 to the sign extended output of adders 819 and 821. The outputs of the accumulators 833 and 835 are then written back to the target registers 837 and 839, and the XPR registers 843 and 844, such that registers 843 and 837contain one of the 40 bit results and registers 844 and 839 contain the other 40 bit result. Real and imaginary results are specified by instructions.

Fig. 9 shows an integrated product adder and accumulator (IPAA) 900 in accordance with the present invention. IPAA 900 may be suitably utilized with hardware 800, replacing an adder and accumulator, to decrease delay and improve performance. For instructions not requiring an accumulated result, select signal 902 controls multiplexer 904 to input a zero value 910 to IPAA 900 which performs addition or subtraction on product operands 906 and 908. For instructions requiring an accumulated result, select signal 902 controls multiplexer 904 to input an accumulated input 912 to IPAA 900 which performs addition or subtraction on product operands 906 and 908 to produce an accumulated result.

Fig. 10 shows a high level view of a hardware apparatus 800' suitable for implementing the multiply complex instructions for execution in a single cycle of operation. Hardware apparatus 800' includes many of the same elements as hardware apparatus 800, with common elements to both embodiments designated by the same element numbers. The multiplier alignment multiplexer 823 and registers 824a, 824b, 824c and 824d of apparatus

800 are replaced by a logical array 850, allowing the multiply complex instructions to complete in a single cycle of operation. The logical array 850 properly aligns the outputs of multipliers 807, 809, 811 and 813 for transmission to the adders 819 and 821.

Computation of a Covariance Matrix

The multiply complex long instructions of the present invention may be advantageously used in the computation of a covariance matrix. As an example, consider an antenna array consisting of several elements arranged in a known geometry. Each element of the array is connected to a receiver that demodulates a signal and produces a complex-valued output. This complex-valued output is sampled periodically to produce a discrete sequence of complex numbers. The elements from this sequence may be organized into a vector of a certain length, called a frame, and may be combined with the vectors produced from the remainder of the antenna elements to form a matrix.

For an antenna array with M elements and K samples per frame, a matrix U is created.

$$U_{M\times K} = \begin{bmatrix} [u_0(0) & u_0(1) & \cdots & u_0(K-1)] \\ [u_1(0) & u_1(1) & \cdots & u_1(K-1)] \\ & \vdots & & \\ [u_{M-1}(0) & u_{M-1}(1) & \cdots & u_{M-1}(K-1)] \end{bmatrix}$$

$$R_{MxM} = U \times U^H$$

In problems such as direction of arrival algorithms, it is necessary to compute the covariance matrix from such received data. For zero-mean, complex valued data, the covariance matrix, R, is defined to be

where '^{H}' is the *hermitian* operator, denoting a complex conjugate matrix transpose.

For example, assuming M=12 and K=128, the elements of R are computed as $R_{i,j} = \sum_{k=0}^{K-1} u_i(k) \times (u_j(k))^*$, which corresponds to the summation of 128 complex conjugate multiplies for each of the 144 elements of R. As seen in Fig. 11A, R is a 12x12 matrix 1100. R is conjugate-symmetric, so the upper triangular portion of R is the complex conjugate of the lower triangular portion. $R_{i,j}=R_{j,i}^*$ for $i\neq j$. As seen in Fig. 11B, this symmetry allows an optimization such that only 78 elements of R, the lower triangular portion and the main diagonal, need to be computed, as the remaining elements are the conjugated copies of the lower diagonal.

Each element in U is represented as a 16-bit, signed (15 information bits and 1 sign bit), complex value (16-bit real, 16-bit imaginary). Fixed-point algebra shows that the

multiplication of two such values will result in a complex number with a 31-bit real and 31-bit imaginary component (30 information bits and 1 sign bit). The accumulation of 128 31-bit complex numbers, to avoid saturation (achieving the maximum possible positive or minimum possible negative value available for the given number of bits), requires 39 bits of accuracy in both real and imaginary components (38 information bits and 1 sign bit). Therefore to compute the covariance matrix for this system, it is necessary to utilize the complex multiply-accumulate function that achieves 31 complex bits of accuracy for the multiply, and can accumulate these values to a precision of at least 39 complex signed bits.

The computation of the 78 elements of the covariance matrix 1100 may be advantageously accomplished with the ManArray 2x2 iVLIW SIMD processor 100 shown in Fig. 1. Utilizing the single cycle pipeline multiply complex conjugate long with extended precision accumulate (MPYCXJLXA) instruction described above, 128 complex multiplies can be executed in consecutive cycles. As the iVLIW processor 100 allows 64 bits to be loaded into each PE per cycle, the computation of a single length 128 complex conjugate dot product is accomplished in 130 cycles, for a 2 cycle MPYCXJLXA. For a single cycle MPYCXJLXA, the computation is performed in 129 cycles.

Figs. 11C-11I show the computations performed by the 4 PEs (PE0, PE1, PE2 and PE3) of processor 100 to calculate the 78 elements of the covariance matrix R 1100. As seen in Fig. 11C, for iteration 1 PE0 performs the multiplications for $R_{0,0}$, PE1 performs the multiplications for $R_{1,1}$, PE2 performs the multiplications for $R_{2,2}$, and PE3 performs the multiplications for $R_{3,3}$. As seen in Fig. 11D, for iteration 2 PE0 performs the multiplications for $R_{4,4}$, PE1 performs the multiplications for $R_{5,5}$, PE2 performs the multiplications for $R_{6,6}$, and PE3 performs the multiplications for $R_{7,7}$. As seen in Fig. 11E, for iteration 3 PE0 performs the multiplications for $R_{8,8}$, PE1 performs the multiplications for $R_{9,9}$, PE2 performs the multiplications for $R_{10,10}$, and PE3 performs the multiplications for $R_{11,11}$. Figs. 11F-H show the multiplications for iterations 4-11, 12-15, 16-18 and 19-20, respectively. Thus, the computation of the 78 elements of the covariance matrix from a 12x128 data matrix of 16-bit signed complex numbers occurs in 20 (dot product iterations) x 130 (cycles per dot product) = 2600 cycles, plus a small amount of overhead. The remaining elements of R are simply the conjugated copies of the lower diagonal. Prior art implementations typically would consume 79,872 cycles on a single processor with 8 cycles per complex operation, 128 complex operations per dot product and 78 dot products.

While the present invention has been disclosed in the context of various aspects of presently preferred embodiments, it will be recognized that the invention may be suitably applied to other environments consistent with the claims which follow.